Module 1: Authentication & User Profile

Author: Parv Bansal
UID – 23BCS13701
Section – Krg 2-A

1. Objective

The objective of this module is to provide a secure and efficient authentication and profile management system for users in a financial or personal data management application. It ensures that only authorized users can access their respective data while protecting all user credentials and sensitive information. This system uses robust techniques such as password hashing, token-based authentication, and secure session handling to prevent unauthorized access and data breaches.

2. Features

The Authentication and User Profile Module consists of the following key features that collectively ensure data integrity, user privacy, and smooth access control:

• User Registration & Login: Allows users to register using email/username and password. Passwords are securely hashed before storing in the database. Login functionality verifies credentials and issues authentication tokens.

• JWT Authentication with Refresh Tokens: Implements stateless authentication using JSON Web Tokens (JWT). A short-lived access token is issued for quick authentication, and a long-lived refresh token allows re-issuance of access tokens without re-login.

• Password Reset via Email/OTP: Provides users with the ability to reset forgotten passwords using a secure OTP or reset link sent via email.

• Profile Management: Enables users to view and update personal details such as name, currency, locale preferences, and notification settings through authenticated API calls.

• Role-Based Access (User/Admin): Ensures access control by assigning roles to users. Admins can manage other users while normal users can only access their own data.

3. Technical Flow

The following steps outline the technical workflow from user registration to profile management:

1. User signs up by providing name, email, and password. Backend validates input and stores the hashed password in the database.

2. On successful login, the backend verifies credentials and issues a JWT (valid for 1 hour) along with a Refresh Token (valid for 7 days).

3. The frontend securely stores JWT tokens, preferably in HTTP-only cookies or encrypted storage.

4. For each API call, an authentication middleware verifies the token before allowing access to protected endpoints.

5. The Profile Management API allows authenticated users to fetch and update their personal information.

4. Sample Implementation (Node.js / Express)

The following code demonstrates the implementation of authentication and profile management using Node.js, Express, MongoDB (via Mongoose), bcrypt for password hashing, and JWT for token generation.

```
// authUserProfileModule.js
import express from "express";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
import User from "../models/User.js";
import { authenticate } from "../middleware/auth.js";

const router = express.Router();

// ============================
// REGISTER NEW USER
// ============================
router.post("/register", async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const hashed = await bcrypt.hash(password, 10);
    const user = new User({ name, email, password: hashed });
    await user.save();
    res.status(201).json({ message: "User registered successfully" });
  } catch (err) {
    res.status(500).json({ error: "Registration failed" });
  }
});

// ============================
// LOGIN USER
```

```
// ============================
router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) return res.status(404).json({ error: "User not found" });

    const valid = await bcrypt.compare(password, user.password);
    if (!valid) return res.status(401).json({ error: "Invalid password" });

    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
expiresIn: "1h" });
    const refreshToken = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
expiresIn: "7d" });

    res.json({ token, refreshToken });
  } catch (err) {
    res.status(500).json({ error: "Login failed" });
  }
});

// ============================
// FETCH PROFILE
// ============================
router.get("/profile", authenticate, async (req, res) => {
  const user = await User.findById(req.user.id).select("-password");
  res.json(user);
});

// ============================
// UPDATE PROFILE
// ============================
router.put("/profile", authenticate, async (req, res) => {
  const updated = await User.findByIdAndUpdate(req.user.id, req.body, {
new: true });
  res.json(updated);
});

export default router;
```

---

5. Authentication Middleware (auth.js)

The middleware is responsible for validating JWT tokens included in API request headers. It ensures that only users with valid tokens can access protected endpoints.

```
import jwt from "jsonwebtoken";

export const authenticate = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader) return res.status(401).json({ error: "No token provided"
});

  const token = authHeader.split(" ")[1];
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(403).json({ error: "Invalid or expired token" });
  }
};
```

---

6. Detailed Explanation of Code

1. The `/register` route handles user registration by hashing passwords using bcrypt and storing them securely.
2. The `/login` route authenticates users by comparing provided credentials with the database records and issues JWTs.
3. The `/profile` (GET) route retrieves the user's profile, excluding password details.
4. The `/profile` (PUT) route allows authenticated users to update their information.
5. The `authenticate` middleware checks for valid JWT tokens before allowing access to secure routes.

7. Conclusion

The Authentication and User Profile Module serves as the backbone of user security and data privacy in web applications. By combining JWT authentication, secure password storage, and role-based access control, it establishes a robust system for protecting sensitive user information while offering flexibility and scalability for future enhancements.